Types of expressions

An expression describes a computation and evaluates to a value

\[
\begin{align*}
18 + 69 & & \sin \pi & & \log_2 1024 \\
6 & & \frac{6}{24} & & \sqrt{4093}101 \\
23 & & 7 \text{ mod } 2 & & \lim_{x \to -1} \frac{1}{x + 2} \\
1869 & & -1869 & & (69)^\frac{1}{18}
\end{align*}
\]

Anatomy of a Call Expression

All expressions can use function call notation

(Demo)

Evaluation procedure for call expressions:
1. Evaluate the operator and then the operand subexpressions
2. Apply the function that is the value of the operator to the arguments that are the values of the operands
Names, Assignment, and User-Defined Functions

Types of Expressions

Primitive expressions:
- 2
- add
- ‘hello’

Call expressions:
- max(2, 3)

Operator
Operand
Operand

An operand can also be a call expression:
max(min(5, 1), -4), min(2, -2)

Discussion Question 1
What is the value of the final expression in this sequence?
```python
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
>>> max(f(2, g(h(1, 5), 3)), 4)
```

Environment Diagrams

Environment diagrams visualize the interpreter’s process.

Assignment Statements

Execution rule for assignment statements:
1. Evaluate all expressions to the right of = from left to right.
2. Bind all names to the left of = to those resulting values in the current frame.

Discussion Question 1 Solution

```python
1 f = min
2 f = max
3 g, h = min, max
4 max = g
5 (max((f(2, g(h(1, 5), 3)), 4)
```

Defining Functions
Defining Functions

Assignment is a simple means of abstraction: binds names to values.
Function definition is a more powerful means of abstraction: binds names to expressions.

```
>>> def square(formal_parameters):
    return return_expression
```

Function body defines the computation performed when the function is applied.

Execution procedure for def statements:
1. Create a function with signature <name>(<formal parameters>)
2. Set the body of that function to be everything indented after the first line
3. Bind <name> to that function in the current frame

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):
1. Add a local frame, forming a new environment
2. Bind the function’s formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):
1. Add a local frame, forming a new environment
2. Bind the function’s formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.
So far, the current environment is either:
• The global frame alone, or
• A local frame, followed by the global frame.

Most important two things I’ll say all day:
An environment is a sequence of frames.
A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
E.g., to look up some name in the body of the square function:
• Look for that name in the local frame.
• If not found, look for it in the global frame.
(Built-in names like “max” are in the global frame too, but we don’t draw them in environment diagrams.)